ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA161

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

APPLYING MODEL CHECKING TO CRITICAL PLC APPLICATIONS: AN ITER CASE STUDY

B. Fernández*, D. Darvas, E. Blanco, CERN, Geneva, Switzerland
Gy. Sallai, BME, Budapest, Hungary
I. Prieto, IBERINCO, Madrid, Spain
G. Lee, Mobiis Co. Ltd., Seoul, South Korea
B. Avinashkrishna, Y. Gaikwad, S. Sreekuttan, Tata Consultancy Services, Pune, India
R. Pedica, Vitrociset s.p.a, Rome, Italy

Abstract

The development of critical systems requires the application of verification techniques in order to guarantee that the requirements are met in the system. Standards like IEC 61508 provide guidelines and recommend the use of formal methods for that purpose. The ITER Interlock Control System has been designed to protect the tokamak and its auxiliary systems from failures of the components or incorrect machine operation. ITER has developed a method to assure that some critical operator commands have been correctly received and executed in the PLC (Programmable Logic Controller). The implementation of the method in a PLC program is a critical part of the interlock system. A methodology designed at CERN has been applied to verify this PLC program. The methodology is the result of 5 years of research in the applicability of model checking to PLC programs. A proof-of-concept tool called PLCverif implements this methodology. This paper presents the challenges and results of the ongoing collaboration between CERN and ITER on formal verification of critical PLC programs.

INTRODUCTION

ITER aims to be the first fusion device that produces net energy. To achieve this goal, thousands of engineers work on building the world's largest tokamak to prove the feasibility and pave the path to commercial production of fusion-based electricity. This unique installation has many associated safety risks and the Interlock Control System is in charge of the supervision and control of all the ITER components involved in the instrumented protection of the tokamak and its auxiliary systems. This system implements the interlock functions to protect ITER from incorrect operation and other hazards. The architecture of the Interlock Control System is based on Programmable Logic Controllers (PLCs) on the control layer and WinCC OA SCADA (Supervisory Control and Data Acquisition) on the supervision layer..

Under certain exceptional situations, like commissioning or maintenance, the interlock functions have to be disabled or some interlock signals shall be masked or forced in the interlock control system. This has to be performed remotely, via the SCADA system. The SCADA system also ensures that the operator will always be aware of the presence of any active masked interlock function. The ITER developers have designed and implemented the HIOC (High Integrity Operator Commands) protocol to ensure that these critical commands sent from SCADA are properly received by the PLCs.

The PLC code that implements the HIOC mechanism is a critical part of the system. For this reason, in addition to the traditional testing techniques, model checking has been applied to verify the behaviour of the program according to the given specifications. Model checking is a formal verification method that takes the model of a system and a formalised requirement, and checks whether the requirement is satisfied by the modelled system with mathematical precision.

Motivation

The ultimate goal of the presented work is to verify the PLC program implementing the HIOC protocol, to prove that it satisfies its specification. The verification project is still in progress, however, we can already report on our experience about how formal verification can reveal implementation faults, flaws in the protocol, or to help understanding the precise requirements.

The goal of this CERN-ITER collaboration is to apply the novel verification technologies developed at CERN to improve the reliability of the HIOC protocol's PLC implementation to be used by ITER.

Related Work

The IEC 61508 standard on functional safety provides guidance for developing an application and communication between the components. The protocols available in WinCC OA for communication with the PLCs do not provide the required level of integrity. The standards, such as IEC 61784, discuss the use of "black channels" for critical applications, i.e. providing a safe communication channel by adding various countermeasures without depending on the reliability of the underlying (non-safe) channel. The HIOC protocol follows this philosophy.

Formal verification of PLC programs is not part of the industrial state-of-the-practice yet. However, more and more projects aim to support and improve the development of industrial control software by complementing the traditional methods with formal verification [1, 2]. CERN is committed to push the affordable formal verification of PLC programs forward, which lead already to successful verification case

* Corresponding author. E-mail: borja.fernandez.adiego@cern.ch

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA161

studies where widely-used PLC framework block libraries [3] or logics of safety PLCs [4] were verified.

HIGH-INTEGRITY COMMUNICATION PROTOCOL

The goal of the HIOC communication protocol is to provide a safe, reliable communication between the WinCC OA SCADA and the Siemens PLCs for overriding certain signals or interlocks. In general terms, this protocol transfers the information whether a given signal shall be masked or not. In the design of the protocol, the communication medium is regarded as a black channel (according to the IEC 61508-2 standard), meaning that no guarantee can be provided for the communication channel, moving the responsibility for the safety integrity to the PLC and the operator. The latter is responsible to manually verify the codes presented on the SCADA screens based on written operation procedures.

Mechanism. Each PLC is responsible for one Boolean signal (e.g. an interlock or the position request of a valve) that can be overridden, i.e. a pre-defined value can be forced instead of the value computed by the logic. The PLCs are identified by their unique controller IDs.

The HIOC protocol is a three-step communication protocol between the PLCs and the SCADA, meaning that a successful communication sequence consists of three messaging steps. In each step, first, the SCADA sends a message to the PLC, then the PLC responds to the SCADA. The different steps of the communication are identified by a number, called flag, included in the messages.

A message between the SCADA and the PLC (independently from the direction) consists of three numbers:

- The *controller ID*, identifying the PLC targeted by the action,
- A *flag*, identifying the step of the communication sequence, and
- A *message ID*, identifying the signal to be overridden (or released).

Each override command has three identifiers defined. These identifiers shall be sent in the correct order (each of them corresponds to one of the steps) to successfully override a signal. These identifiers are included in the message ID field of the message. Similarly, three identifiers are defined to release an override. The PLC will successfully override a signal if the following conditions are satisfied:

- The controller ID matches the controller ID of the current PLC in all three messages,
- The flags correspond to the successive steps of the protocol, in the correct order (denoted in this paper by *STEP1*, *STEP2*, *STEP3*), and
- The message IDs match the three defined identifiers for overriding (releasing), matching the step described by the flag (denoted in this paper by *OVR1*, *OVR2*, *OVR3* for overriding).

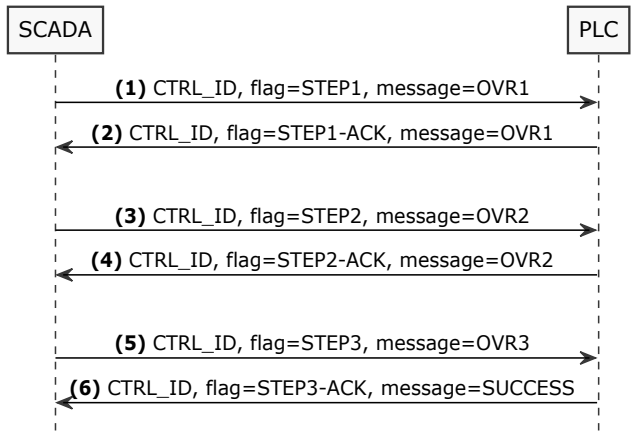

Figure 1: Communication sequence successfully overriding a signal

Furthermore, a timer is also included in the PLC program to ensure that the verification process is only successful if it is completed within a predefined time window.

This communication sequence is illustrated in Figure 1.

If a message received by a PLC contains a wrong controller ID, the message is discarded. If the flag or the message identifier does not match the expected next value, the current communication sequence is aborted and the SCADA is notified about this.

Specification. The requirements and high-level architecture of the HIOC protocol is defined in a textual format, with flowcharts and detailed examples for the key parts of the protocol. This informal specification was the main source of the requirements to be verified. However, due to the inherent ambiguity of natural text and also because the specification does not discuss the low-level PLC-specific implementation details, an intensive discussion between the designers of the protocol, specifiers and verifiers was required.

Implementation. The target of this verification is the PLC program of the HIOC protocol, thus we introduce this part in more detail. Due to various constraints, the HIOC was designed as an interconnection of two blocks: a fail-safe function block (`HIOC_SFT`) written in Siemens CFC (Continuous Function Chart), and a standard function block (`HIOC_STD`) written in Siemens SCL. The `HIOC_STD` block is responsible for handling the messages received from the SCADA and to send the appropriate responses. The fail-safe `HIOC_SFT` block performs the comparison of the expected and received values, and computes whether the signal shall be masked or not, based on the communication in the past.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA161

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

SYSTEM MODELLING AND REQUIREMENTS FORMALISATION

This section overviews the workflow and toolchain used for the formal verification of the HIOC protocol, and the details of modelling and requirement formalisation.

PLCverif: Model Checking Methodology for PLC Programs

PLCverif is a tool for formal verification of PLC programs [5]. The methodology implemented by PLCverif is designed to be general so several PLC languages and verification tools can be included in it. The verification workflow (shown in Figure 2) is based on the source code of PLC programs. Currently, the Siemens SCL and STL languages are supported by PLCverif. Using the export capabilities of the Siemens development environments, programs written in CFC, LAD and FBD languages can also be checked.

The requirements are described either using requirement patterns or assertions. A *requirement pattern* is a fixed English sentence with certain placeholders to be filled by the user. An *assertion* in this context is a Boolean expression included in the source code as a special comment. This expression shall be true every time the given part of the program is executed.

The source code of the PLC program is translated into an *intermediate representation*, based on control flow graphs (CFG). This intermediate representation – together with the requirements formalised using the patterns – can then be used to generate automaton-based representation of the PLC program for various general-purpose model checkers, such as nuXmv. Alternatively, the intermediate representation can be the source for generating structured programs for software model checkers [6], such as CBMC that works on annotated C code and checks whether all assertions are always satisfied in the code. This alternative was recently added to the methodology to evaluate the performance of software model checkers for PLC programs.

Based on the PLC code and the requirements provided by the user, PLCverif will automatically generate and reduce the necessary intermediate models, execute the chosen external verification tools, and provide a verification report. This verification result will describe whether the given requirement is satisfied by the checked program, and if not, an understandable counterexample is also provided, exemplifying a violation of the requirement.

PLC Program Modelling

As previously discussed, the HIOC protocol's PLC implementation consists of two interconnected blocks: `HIOC_SFT` and `HIOC_STD`. Both were given in SCL format for verification. Parsing `HIOC_STD` in PLCverif did not pose any major challenge, however the `HIOC_SFT` block, originally written in CFC relies on many external function blocks, such as logic operation blocks (e.g. `AND4`), comparison blocks (e.g. `F_CMP_R`), data conversion blocks (e.g. `F_R_FR`), flip-flops (e.g. `F_RS_FF`), etc. These blocks (or their simplified version in some cases) had to be re-implemented in SCL based on the documentation available from Siemens [7].

After having the complete implementation parsed, the whole program is automatically translated either to an automaton-based intermediate format that is later suitable for general-purpose model checking (e.g. by using the nuXmv tool), or a verification-oriented C representation can also be generated that can be checked by specialised software model checker tools (such as CBMC).

Depending on the requirements, the user may want to analyse only the `HIOC_SFT` block, or only the `HIOC_STD` block independently. In other cases, the two blocks shall be analysed together, as they are connected in the real implementation. As both the `HIOC_SFT` and `HIOC_STD` blocks have many inputs and outputs to connect to the other block, checking the two blocks together as a system may be easier due to the reduced number of requirements. In addition, as the specification is written for the two blocks together, specifying their joint behaviour is more feasible, thus most of the verifications were performed on the two interconnected blocks.

The SCL representation of the two blocks and their interconnections is about 1,500 lines long, containing more than 150 variables, out of which about 50 are non-Boolean variables.

Requirement Formalisation

To formalise the requirements, we have used both assertions and requirement patterns. Assertions can be regarded as special, simple requirement patterns. The main practical difference is that as the requirement patterns have a higher expressive power, certain model checkers (e.g. CBMC) can only be used to check assertions, not requirement patterns.

In some cases, we have formalised non-existent requirements to gain more knowledge about the implementation via the provided counterexamples. For example, the requirement *"It is impossible to successfully override a given function."* is obviously not expected to be satisfied. However, the violation described by the counterexample is an example (so-called witness) of a successful interlock masking.

VERIFICATION RESULTS

This section presents two representative verification cases to illustrate how model checking helped to improve the quality of the HIOC implementation and the understanding of its specification.

Pattern-based verification. The first representative verification case is meant to check that a given step of the HIOC protocol is executed correctly. In this case the checked property expressed that the response to a message with *flag=STEP1* shall be *flag=STEP1-ACK*, provided that the received message is the expected *message=OVR1*, as shown in Figure 1. Obviously, similar verification cases can check the correct behaviour in case of receiving incorrect messages (when

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA161

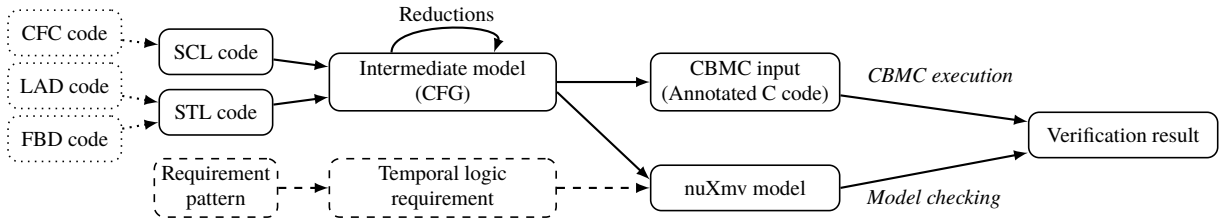

Figure 2: Formal verification workflow of PLCverif

Table 1: Counterexample

Variable	PLC Cycle 1	PLC Cycle 2
Inputs		
CTRL_ID (SCADA to PLC)	*IDLE*	*ID1*
FLAG (SCADA to PLC)	*IDLE*	*STEP1*
MESSAGE (SCADA to PLC)	*IDLE*	*OVR1*
fault1	FALSE	FALSE
fault2	FALSE	FALSE
enabled	FALSE	TRUE
parameter1	FALSE	FALSE
parameter2	FALSE	TRUE
parameter3	TRUE	TRUE
Outputs		
FLAG (PLC to SCADA)	*IDLE*	*IDLE*

the communication sequence shall be aborted), or to check behaviour in later steps.

This requirement was formalised using the verification patterns provided by PLCverif, and the verification was done using the nuXmv tool. The first verification results were obtained in only 29 seconds. However, this needed both the automated reductions built in PLCverif, and some manual simplifications of the code under verification (e.g. reducing the size of integer data types).

Even though the requirement was expected to be satisfied by the developers, the verification report provided by PLCverif clearly showed that the PLC program was not complaint with the specification. We have quickly realised that the requirement did not consider the case when the PLC receives – and discards – a message that targets another PLC. We have also found out that this expected, general behaviour can be altered by various communication fault signals. Also, the override protocol can be disabled, in which case the requirement does not have to be satisfied. While so far this did not reveal any implementation faults, it helped the specifiers and verifiers to understand more deeply the requirements and the various corner cases of the protocol.

After excluding all these special behaviours, the verification of the formalised requirement still showed a potential violation. The counterexample for this violation is illustrated in Table 1. The values in italic are symbolic values for illustration purposes. The real variable names were altered to facilitate the understanding. The root cause of this deviation is currently under evaluation by the domain experts. If it reveals a new special behaviour that needs to be excluded, the verification will continue with a more precise formal requirement.

Using similar approach, we were able to identify various discrepancies between the implementation and the specification in earlier stages of the HIOC protocol development, which were consequently fixed.

Assertion-based verification and witness generation. The other main group of verification cases used simpler requirements (formalised using assertions), however more precise, unreduced representation of the PLC program. Some of the included assertions were expressing real requirements, such as the signal can only be overwritten if a "successful overriding" message was sent before by the PLC. Other assertions were only used to generate witnesses of the key functionality, as discussed earlier.

The assertions were included in the SCL code, then PLCverif generated an annotated C code representation. This was fed into the CBMC verification tool, which was able to generate counterexamples in a couple of seconds, even though no reductions were performed on the PLC program or the intermediate representation in this case. The counterexamples produced in this case are similar to the one presented in Table 1, however they typically involve several consecutive PLC cycles, thus the table contains more columns with different variable values.

These counterexamples demonstrated the possibility of various unexpected execution traces. Some of them showed the possibility of overriding the critical signals even if the some of the SCADA to PLC messages were not received in correct order, or if they were overlapped by other, potentially erroneous messages. Other counterexamples indicated the possibility of ignoring an abort requested on the SCADA side, and still perform the initiated operation.

It has to be noted that the SCADA implementation of the HIOC protocol was not part of the verification, thus the various checks implemented in the SCADA layer are not considered. However, this is consistent with the black channel assumption. Now it is up to the specification owners and developers to analyse the counterexamples and to decide whether they are (1) manifestations of real development faults, (2) caused by incorrect or incomplete requirements and/or specification, or (3) due to imprecisions in the verification workflow.

In case of non-trivial protocols, such as HIOC, verification of correctness using only assertions does not seem to be feasible. To describe the complete set of requirements, the larger expressiveness of requirement patterns looks necessary. However, this method is an easy-to-use, inexpensive

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA161

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

way to generate interesting execution traces that can reveal unintended behaviours.

CONCLUSIONS

Although the complete formal verification of the HIOC protocol is still in progress, model checking has shown that it can be a valuable contribution to the general toolset of industrial controls engineers and PLC developers. This is especially true for safety-critical uses, where the correct behaviour is extremely important.

Formal verification aids the development process in various ways:

- First of all, a *formal proof of correctness* ensures the lack of development faults. It is typically difficult to achieve that level due to various reasons. As we are relying on various external tools (model checkers, compilers, development frameworks, etc.), also due to the potential mistakes in understanding the requirements, the absolute certainty of correctness is very rarely achievable. In case of the HIOC protocol, we are not claiming yet complete absence of development faults, however we plan to continue this work until all our formalised requirements become satisfied.

- Model checking can help to *better understand the program under development*. Even if a requirement does not express some behaviour that must be satisfied, the counterexample given to it may help to explore interesting behaviours. For example, checking that the HIOC protocol *cannot* override a signal successfully (that is obviously not a real requirement), the counterexamples will show evidences how the signal can be masked. By constraining the counterexample (i.e. changing the requirement such that the trivial violations are excluded) interesting, peculiar behaviours can be found.

- Last, the need to formally express the requirements inherently helps to understand the requirements precisely. Very often model checking necessitates a long discussion between the specifier, developer and verifier. Model checking can reveal that a requirement that sounds sensible may exclude many of the corner cases, and these have to be gradually understood by all parties involved.

The verification of the HIOC protocol revealed suspicious corner cases where the implementation does not match the specification. Now it is the duty of the domain experts and developers to understand the implications and to improve the implementation or to make the specification more precise. We are going to continue this work until the results are convincingly demonstrating the good quality and correct behaviour of the HIOC protocol.

REFERENCES

[1] S. Biallas, J. Brauer, and S. Kowalewski, "Arcade.PLC: A verification platform for programmable logic controllers," in *Proc. 27th IEEE/ACM Int. Conf. on Automated Software Engineering*. IEEE, 2012, pp. 338–341.

[2] T. Lange, M. R. Neuhäußer, and T. Noll, "Speeding up the safety verification of programmable logic controller code," in *Hardware and Software: Verification and Testing*, ser. LNCS. Springer, 2013, vol. 8244, pp. 44–60.

[3] B. Fernández, D. Darvas, J.-C. Tournier, E. Blanco, and V. M. González, "Bringing automated model checking to PLC program development – A CERN case study," in *Proc. 12th Int. Workshop on Discrete Event Systems*, ser. IFAC Proceedings Volumes, vol. 47 (2). Elsevier, 2014, pp. 394–399.

[4] D. Darvas, I. Majzik, and E. Blanco, "Formal verification of safety PLC based control software," in *Integrated Formal Methods*, ser. LNCS. Springer, 2016, vol. 9681, pp. 508–522.

[5] D. Darvas, B. Fernández, and E. Blanco, "PLCverif: A tool to verify PLC programs based on model checking techniques," in *Proc. 15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems*. JACoW, 2015, pp. 911–914.

[6] G. Sallai, D. Darvas, and E. Blanco, "Testing, simulation, and visualisation of PLC programs using x86 code generation," CERN, Techical report EDMS 1844850, 2017. [Online]. Available: http://edms.cern.ch/document/1844850

[7] Siemens, *SIMATIC Industrial Software S7 F/FH Systems – Configuring and Programming*, 2009. [Online]. Available: http://support.industry.siemens.com/cs/document/2201072